Symmetry-reinforced Nogood Recording from Restarts

Christophe Lecoutre, Sebastien Tabary

Symmetry-reinforced Nogood Recording from Restarts

Christophe Lecoutre and Sébastien Tabary

CRIL - CNRS, UMR 8188,
Univ Lille Nord de France, Artois
F-62307 Lens, France
{lecoutre,tabary}@cril.fr

Abstract. Nogood recording from restarts is a form of lightweight learning that combines nogood recording with a restart strategy. At the end of each run, nogoods are extracted from the current (rightmost) branch of the search tree. These nogoods can be used to prevent parts of the search space from being explored more than once. In this paper, we propose to reinforce nogood recording (from restarts) by exploiting symmetries: every time the solver has to be restarted, not only the nogoods that are extracted from the current branch are recorded, but also some additional nogoods that can be computed by means of the previously identified problem symmetries. This mechanism of computing symmetric nogoods can be iterated until a fixed-point is reached, and controlled (if necessary) by limiting the number and/or the size of recorded nogoods.

1 Introduction

The runtime distribution of a randomized search algorithm is sometimes characterized by an extremely long tail with some infinite moment (e.g., see [14]). For some constraint satisfaction (optimization) problems, it has been found worthwhile to employ restarts with a randomized search heuristic. However, if restarts are employed without learning, the average performance of the solver can be damaged on some problem instances because the same parts of the search space may be explored several times. At the opposite, nogood recording without restarts has not yet been shown to be entirely convincing for constraint satisfaction; when uncontrolled, nogood recording can lead to exponential space complexity. Although restarts without nogood recording, and also nogood recording without restarts, may be of limited use, a combination of both of these techniques happen to be more useful. For example, in [18], it has been shown that so-called reduced nld-nogoods can be extracted (to be recorded in a nogood base) from the current (rightmost) branch of the search tree at the end of each run. These nogoods can be used to prevent parts of the search space from being explored more than once. Thus we can then benefit from restarts and learning capabilities without sacrificing solver performance or space complexity.

On the other hand, symmetry breaking is an important research topic in constraint programming. The use of symmetries in search problems is conceptually simple. If two distinct nodes in a search tree are related by a symmetry,

there is no need to explore both of them because symmetries preserve satisfiability. When a node is an internal dead-end, the nodes that are symmetrical to it are guaranteed to be internal dead-ends as well. When a node is the root of a fruitful subtree (i.e., a subtree containing solutions), symmetrical solutions can be computed automatically, i.e. without exploring symmetrical nodes. Breaking symmetries can facilitate determining the satisfiability of an instance or counting/computing the full set of solutions.

Symmetry breaking involves two distinct issues. First, symmetries must be identified. Either the user is asked to perform this (often difficult) task, or otherwise an automatic procedure identifies symmetries. Second, the symmetries must be exploited. For this, there are two main categories of approaches (apart from reformulation techniques). One approach posts symmetry-breaking constraints during a preprocessing stage, to speed up subsequent search. The other main strategy is to use symmetries dynamically during search to prevent exploration of irrelevant nodes. For the exploitation of symmetries, published methods include symmetry-breaking constraints [6, 10, 26], symmetry-breaking heuristics [20], symmetry breaking during search (SBDS) [1, 13, 12], symmetry breaking via dominance detection (SBDD) [9, 11, 23, 24], amongst others.

In this paper, we propose to reinforce nogood recording (from restarts) by exploiting problem symmetries. The principle is quite simple: every time the solver has to be restarted, not only the nogoods that are extracted from the current branch are recorded, but also some additional nogoods that can be computed by means of the problem symmetries. Interestingly, this mechanism of computing symmetric nogoods can be iterated until a fixed-point is reached. Of course, if necessary, one can control this form of reinforced learning by limiting the number of recorded nogoods and/or their size.

2 Technical Background

A *constraint network* (CN) P is composed of a finite set of n variables, denoted by $vars(P)$, and a finite set of e constraints, denoted by $cons(P)$. Each variable x has a domain which is the finite set of values that can be assigned to x. The initial domain of a variable x is denoted by $dom^{init}(x)$ whereas the current domain of x (in the context of P) is denoted by $dom^P(x)$, or more simply $dom(x)$; we always have $dom(x) \subseteq dom^{init}(x)$. The maximum domain size for a given CN will be denoted by d. A *v-value* is a variable-value pair (x, a) where x is a variable and $a \in dom^{init}(x)$. A v-value *of* a CN P is a v-value (x, a) such that $x \in vars(P)$ and $a \in dom^P(x)$ and $v\text{-}vals(P)$ denotes the set of v-values of P Each constraint c involves an ordered set of variables, called the *scope* of c and denoted by $scp(c)$, and is defined by a relation which is the set of tuples allowed for the variables involved in c. A *unary* (resp., *binary*) constraint involves 1 (resp., 2) variable(s), and a *non-binary* one strictly more than 2 variables.

An *instantiation* I of a set $X = \{x_1, \ldots, x_k\}$ of variables is a set $\{(x_1, a_1), \ldots, (x_k, a_k)\}$ such that $\forall i \in 1..k, a_i \in dom^{init}(x_i)$; X is denoted by $vars(I)$ and each a_i is denoted by $I[x_i]$. An instantiation I *on* a CN P is an instantiation

of a set $X \subseteq vars(P)$; it is *complete* if $vars(I) = vars(P)$. I is *valid* on P iff $\forall (x, a) \in I, a \in dom(x)$. I *covers* a constraint c iff $scp(c) \subseteq vars(I)$, and I *satisfies* a constraint c with $scp(c) = \{x_1, \ldots, x_r\}$ iff (i) I covers c and (ii) the tuple $(I[x_1], \ldots, I[x_r]) \in rel(c)$. An instantiation I on a CN P is *locally consistent* iff (i) I is valid on P and (ii) every constraint of P covered by I is satisfied by I. A *solution* of P is a complete locally consistent instantiation on P; $sols(P)$ denotes the set of solutions of P. An instantiation I on a CN P is *globally inconsistent*, or a *nogood*, iff it cannot be extended to a solution of P.

A *positive decision* δ is a restriction on a variable x of the form $x = a$ whereas a *negative decision* is a restriction of the form $x \neq a$, where $a \in dom^{init}(x)$. When decisions are taken on a CN, we obtain a new CN defined as follows. Let P be a CN and Δ be a set of decisions on P, $P|_\Delta$ is the CN obtained (derived) from P such that, for each positive decision $x = a \in \Delta$, each value $b \in dom(x)$ with $b \neq a$ is removed from $dom(x)$, and, for each negative decision $x \neq a \in \Delta$, a is removed from $dom(x)$. For any set Δ of decisions, $vars(\Delta)$ denotes the set of variables occurring in decisions of Δ. A nogood is defined as a globally inconsistent instantiation. Such nogoods are sometimes said to be *standard* and they correspond to the definition proposed by Dechter in [8]. An alternative definition for standard nogoods is: a standard nogood of P is a set Δ of positive decisions on P such that $P|_\Delta$ is unsatisfiable. Consideration of positive and negative decisions as in [11, 16] leads to a generalization of standard nogoods, called *generalized* nogoods: A set of decisions Δ on a CN P is a *generalized* nogood of P iff $P|_\Delta$ is unsatisfiable. Clearly, a (standard) nogood is generalized but the opposite is not necessarily true. For example, $\Delta = \{x = a, y = b\}$ such that $P|_\Delta$ is unsatisfiable is a standard nogood of P, and consequently by definition a generalized nogood of P. But $\Delta = \{x = a, z \neq c\}$, such that $P|_\Delta$ is unsatisfiable, is a generalized nogood of P which is not standard. In fact a generalized nogood can represent an exponential number of standard ones [16].

The search space of a CN can be reduced by a filtering process (called constraint propagation) based on some properties (called consistencies) that allow us to identify and record explicit nogoods in CNs; e.g., identified nogoods of size 1 correspond to inconsistent values that can be safely removed from variable domains. The most famous consistency is GAC (Generalized Arc Consistency) defined as follows. A value (x, a) of P is *GAC-consistent* iff for each constraint c of P involving x there exists a valid instantiation I of $scp(c)$ such that I satisfies c and $I[x] = a$. P is GAC-consistent iff every value of P is GAC-consistent. For binary constraints, GAC is often referred to as AC (Arc Consistency).

3 Nogood recording from restarts

Henceforward, we consider a backtrack search algorithm using binary branching, taking positive decisions first, maintaining a consistency at each step, and using a restart strategy. For any branch of the search tree (built during a run of the backtrack search algorithm), a set of relevant standard nogoods (instantiations that cannot lead to a solution) can be identified directly. *Nogood recording from*

restarts [18] means recording these nogoods but only for the last (rightmost) branch of the search tree just before the restart.

Each branch of the search tree is a sequence of positive and negative decisions. For each branch starting from the root, a generalized nogood can be extracted from each negative decision [23, 18], as follows. Let $\Sigma = \langle \delta_1, \ldots, \delta_m \rangle$ be a sequence of decisions, if δ_i is a negative decision, with $1 \leq i \leq m$, then the subsequence $\langle \delta_1, \ldots, \delta_i \rangle$ of Σ comprising the i first decisions of Σ is called a *nld-subsequence* (negative last decision subsequence) of Σ. Let P be a a constraint network and Σ be the sequence of decisions taken along a branch (starting from the root) of the search tree built for P. For any nld-subsequence $\langle \delta_1, \ldots, \delta_i \rangle$ of Σ, the set $\Delta = \{\delta_j \in \Sigma \mid 1 \leq j < i\} \cup \{\neg \delta_i\}$ is a generalized nogood of P, called *nld-nogood*.

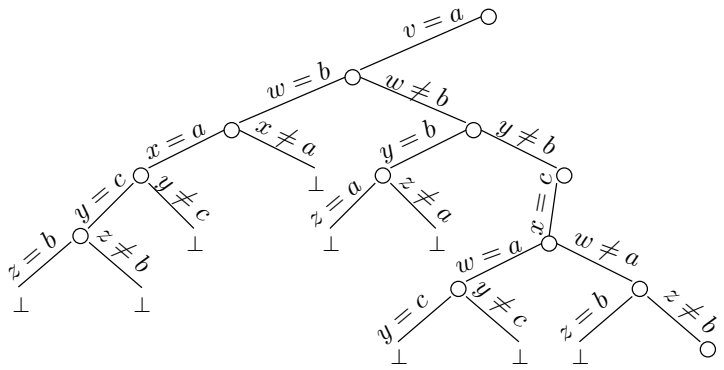


Fig. 1. A partial search tree built by a backtrack search algorithm.

For example, the sequence of decisions taken along the rightmost branch in Figure 1 is $\langle v = a, w \neq b, y \neq b, x = c, w \neq a, z \neq b \rangle$. The nld-subsequences and nld-nogoods that can be extracted from this branch are as follows:

nld-subsequences	nld-nogoods
$\langle v = a, w \neq b \rangle$	$\{v = a, w = b\}$
$\langle v = a, w \neq b, y \neq b \rangle$	$\{v = a, w \neq b, y = b\}$
$\langle v = a, w \neq b, y \neq b, x = c, w \neq a \rangle$	$\{v = a, w \neq b, y \neq b, x = c, w = a\}$
$\langle v = a, w \neq b, y \neq b, x = c, w \neq a, z \neq b \rangle$	$\{v = a, w \neq b, y \neq b, x = c, w \neq a, z = b\}$

Although Σ is a sequence we use set notations ($\delta_j \in \Sigma$) because there is no ambiguity; no decision occurs more than once in Σ. In our particular context, nld-nogoods can be systematically reduced in size by omitting negative decisions, thus reducing space requirements and improving pruning capability. Let P be a constraint network and Σ be the sequence of decisions taken along a branch of the search tree (starting from the root). For any nld-subsequence $\Sigma' = \langle \delta_1, \ldots, \delta_i \rangle$ of Σ, the set $\Delta = pos(\Sigma') \cup \{\neg \delta_i\}$ is a (standard) nogood of P, called a *reduced nld-nogood*, where $pos(\Sigma')$ denotes the set of positive decisions of Σ'.

For example, for the rightmost branch in Figure 1 the nld-nogoods and reduced nld-nogoods are:

nld-nogoods	reduced nld-nogoods
$\{v = a, w = b\}$	$\{v = a, w = b\}$
$\{v = a, w \neq b, y = b\}$	$\{v = a, y = b\}$
$\{v = a, w \neq b, y \neq b, x = c, w = a\}$	$\{v = a, x = c, w = a\}$
$\{v = a, w \neq b, y \neq b, x = c, w \neq a, z = b\}$	$\{v = a, x = c, z = b\}$

The worst-case space complexity to record all reduced nld-nogoods of Σ is $O(n^2d)$. Each nld-nogood is subsumed by its reduced nld-nogood, which has greater pruning capability. Besides, reduced nld-nogood are easier to manage because they are standard nogoods. Indeed, standard nogoods, which are sets (conjunctions) of positive decisions, can be recorded in an equivalent form as sets (disjunctions) of negative decisions. Using this representation, an efficient propagation algorithm can enforce generalized arc consistency on these *nogood constraints* by means of the SAT technique of watched literals [21, 27]. All nogoods are collected in a so-called nogood base denoted here by $\mathscr{B}$.

4 Symmetries on Constraint Networks

The first part of this section provides a brief introduction to *group theory*. A *group* is a pair $(G, \star)$ composed of a set G and a binary operation $\star$ defined on G, such that the following requirements are satisfied:

- Closure: $\forall f \in G, \forall g \in G,\ f \star g \in G$
- Identity element: $\exists e \in G \mid \forall f \in G,\ e \star f = f \star e = f$
- Inverse element: $\forall f \in G, \exists g \in G \mid f \star g = g \star f = e$; g is noted f^{-1}
- Associativity: $\forall f \in G, \forall g \in G, \forall h \in G,\ (f \star g) \star h = f \star (g \star h)$

For example, $(\mathbb{Z}, +)$ is a group, where $\mathbb{Z}$ denotes the set of integers, and $+$ denotes ordinary addition. A *subgroup* of a group $(G, \star)$ is a group $(H, \star)$ such that $H \subseteq G$. The *order* of a group $(G, \star)$ is the number $|G|$ of elements in G. A group can be represented by means of a subset of its elements, called *generating set*. A *generating set* of a group $(G, \star)$ is a subset H of G such that each element of G can be expressed by composition (using $\star$) of elements of H, called *generators*, and their inverses. We write $G =< H >$. A generating set is *irredundant* iff no generator can be expressed by composition of the other generators. Generating sets allow compact representations of groups since for any group $(G, \star)$, there exists a generating set of size $log_2(|G|)$ or smaller.

We are interested in permutation groups because we will be concerned with the symmetries that are *permutations*. A *permutation* on a set D is a bijection σ defined from D onto D. The image of an element $a \in D$ by σ is denoted by a^σ. For example, let $D = \{1, 2, 3, 4\}$ be a set of four integers. A possible permutation σ on D is: $1^\sigma = 2, 2^\sigma = 3, 3^\sigma = 1$ and $4^\sigma = 4$. A permutation can be represented by a set of *cycles* of the form $(a_1, a_2, \ldots, a_k)$ which means that a_i

is mapped to a_{i+1} for $i \in 1..k-1$ and a_k is mapped to a_1. The set of cycles for our permutation σ is $\{(1,2,3),(4)\}$, but in practice, cycles of length one can be omitted because such cycles have no effect, thereby we obtain $\{(1,2,3)\}$. Let D be a set and let σ_1, σ_2 be two permutations on D. The *composition* $\sigma_3 = \sigma_1 \circ \sigma_2$ of σ_1 and σ_2 is defined as follows: $\forall a \in D$, $a^{\sigma_3} = a^{\sigma_1 \circ \sigma_2} = (a^{\sigma_2})^{\sigma_1}$. For the previous example, if $\sigma' = \{(2,4),(1,3)\}$ is a second permutation on D, then $\sigma'' = \sigma' \circ \sigma$ is: $1^{\sigma''} = 4, 2^{\sigma''} = 1, 3^{\sigma''} = 3$ and $4^{\sigma''} = 2$, which gives $\{(1,4,2)\}$ in cyclic form.

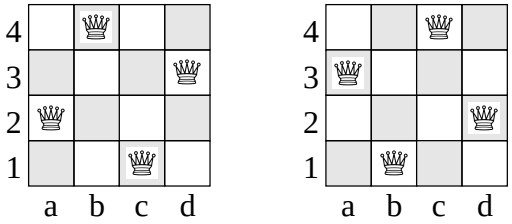
For constraint networks, two definitions of symmetries have been recognized [5] as particularly relevant because they are sufficiently general to encompass most of the previous definitions in the literature. Generality comes from the set on which symmetries are defined: this is the set of v-values (variable-value pairs). The first definition, which is used for example in [17, 22], introduces *solution symmetries* that only preserve sets of solutions. The second definition introduces *constraint symmetries* (or problem symmetries) that preserve the set of constraints. The second definition is less general than the first but is more applicable in practice. A constraint symmetry is a syntactical symmetry [2] that is not limited by necessarily choosing values in the same domain.

A simple mechanism to define symmetries, including solution and constraint symmetries, is to refer to instantiations. Sets of valid instantiations play the role of structure:

Definition 1. *Let P be a constraint network and R be a set of valid instantiations on P. A* symmetry *on P for R is a permutation σ of $v\text{-}vals(P)$ such that $R^\sigma = R$.*

Sometimes symmetries are restricted so that only variables or values are permuted. A *variable symmetry* does not change values whereas a *value symmetry* does not change variables. Let P be a constraint network and R be a set of valid instantiations on P, a *variable symmetry* on P for R is a symmetry σ on P for R such that for every $(x,a) \in v\text{-}vals(P)$, $(x,a)^\sigma = (x^{\sigma_{vars}}, a)$ where σ_{vars} is a permutation of $vars(P)$. Clearly, a variable symmetry is defined (equivalently) by a permutation on $vars(P)$. We shall mainly use this simpler permutation.

To illustrate symmetries on constraint networks, let us consider the 4-queens instance: can we put 4 queens on a board of size 4×4 such that no two queens attack each other? We can model this problem using one variable per queen (column) such that the domains contain values that denote row numbers. Denoting the variables by x_a, x_b, x_c and x_d, to clarify the correspondence with columns, Figure 2 shows the two solutions for this instance. The first is $\{(x_a,2),(x_b,4),(x_c,1),(x_d,3)\}$ and the second is $\{(x_a,3),(x_b,1),(x_c,4),(x_d,2)\}$. This instance has exactly eight constraint symmetries. Among these eight symmetries, if we disregard the identity permutation, only f^h (reflection through the horizontal middle line) is a variable symmetry and only f^v (reflection through the vertical middle line) is a is value symmetry. Using a simplified notation for f^h where only variables are swapped, we obtain in cyclic form $\{(x_a,x_d),(x_b,x_c)\}$, which means that x_a is swapped with x_d and x_b is swapped with x_c.



(a) 1^{st} Solution (b) 2^{nd} Solution

Fig. 2. The two solutions of the 4-queens instance.

5 Symmetry-reinforced Nogood Recording

In this section, we present our approach to reinforce nogood recording from restarts by means of symmetries. First we introduce symmetries on instantiations and nogoods together with symmetry rules that can be connected to earlier works (e.g., [25]), and then we describe an original algorithm to compute symmetrical nogoods. As a related work, the authors of [4] combine lazy clause generation and dynamic symmetry breaking in order to generate symmetric 1UIP (First Unique Implication Point) nogoods. Although we propose here to generate symmetric reduced nld-nogoods at each restart, Chu et al. propose to compute symmetric 1UIP nogoods after each failure during search. Note that the minimization technique presented in [18] would allow us to further improve the quality of nogoods partially in the spirit of the explanations used to compute 1UIPs. The idea of handling symmetric nogoods in the context of restarted solvers has also been studied in [15].

5.1 Symmetries on Nogoods

Because symmetries preserve the structure of constraint networks, we can reason from instantiations and nogoods as explained below. We first need to introduce the notion of *admissible instantiations* [26].

Definition 2 (Admissibility). *Let P be a constraint network, σ be a symmetry on P and I be a valid instantiation on P. I is* admissible *for σ iff I^σ is an instantiation on P.*

For example, consider the instantiation $I = \{(x_a, 1), (x_b, 2)\}$ for the 4-queens instance. For the symmetry r^{90} (rotation by 90° right), we obtain $I^{r^{90}} = \{(x_a, 4), (x_b, 3)\}$, which is an instantiation. Thus I is admissible for r^{90}. Now if $I' = \{(x_a, 1), (x_b, 1)\}$, by r^{90}, we obtain $I'^{r^{90}} = \{(x_a, 4), (x_a, 3)\}$, which is not an instantiation, so I' is not admissible for r^{90}. By definition, an instantiation cannot contain two v-values involving the same variable.

Note that valid instantiations are always admissible for variable symmetries and for value symmetries.

Symmetries preserve solutions. The following proposition is related to Proposition 2.1 in [6].

Proposition 1. *Let P be a constraint network, σ be a symmetry on P and I be a complete valid instantiation on P. I is a solution of P iff I^σ is a solution of P.*

Proof. If $I \in sols(P)$ then necessarily $I^\sigma \in sols(P)$ since $sols(P)^\sigma = sols(P)$. The other direction holds because σ^{-1} is also a solution symmetry.

For example, for the 4-queens instance, $\{(x_a, 2), (x_b, 4), (x_c, 1), (x_d, 3)\}^{f^h}$ gives $\{(x_d, 2), (x_c, 4), (x_b, 1), (x_a, 3)\}$ which is the second solution. Interestingly, this result can be refined as follows.

Proposition 2. *Let P be a constraint network, σ be a symmetry on P and I be a valid instantiation on P. I is a good (globally consistent instantiation) on P iff I^σ is a good on P.*

Proof. If I is a good, this means that there exists at least a solution I' of P that extends I. We know from Proposition 1 that I'^σ is also a solution of P. Necessarily, I'^σ extends I^σ, so I^σ is a good.

Corollary 1. *Let P be a constraint network, σ be a symmetry on P and I be a valid instantiation on P. I is a nogood (globally inconsistent instantiation) on P iff either I is not admissible for σ or I^σ is a nogood on P.*

Proof. From Proposition 2, we know that I is a good on P iff I^σ is a good on P. This is equivalent to: I is not a good on P iff I^σ is not a good on P. Because I is by hypothesis a valid instantiation, I is necessarily a nogood. However, nothing can be said precisely about I^σ. I^σ is not a good, which means that either I is not admissible for σ or I^σ is a nogood.

5.2 An Algorithm to Reinforce Nogood Recording from Restarts

Following [18], when the current run is stopped, the function extractNogoods, Algorithm 1, derives reduced nld-nogoods from the current branch of the search tree. This function admits as a parameter the sequence of decisions taken along the current rightmost branch and returns a set of standard nogoods. Each negative decision in this sequence yields a standard nogood. From the root to the last decision of the current branch, we record successive positive decisions (in a set denoted by *positiveDecisions*). For each negative decision encountered, the algorithm constructs a standard nogood Δ from the negation of this decision and all previous positive decisions recorded (line 7). Computed nogoods are collected in a set Γ. Additionally, for each nogood Δ identified from the sequence of decisions, the procedure computeSymmetricNogoodsOf (Algorithm 2) is called at line 8 to compute a set of symmetrical nogoods from Δ that can be added to Γ. Nogoods in the set Γ computed by extractNogoods will be later added to the global nogood base $\mathscr{B}$, and exploited in subsequent runs.

Algorithm 1: extractNogoods(Σ: sequence of decisions): set of nogoods

Input: $\Sigma = \langle \delta_1, \ldots, \delta_m \rangle$
Output: a set Γ of nogoods, computed from Σ

```
1 Γ ← ∅
2 positiveDecisions ← ∅
3 for i ranging from 1 to m do
4     if δ_i is a positive decision then
5         positiveDecisions ← positiveDecisions ∪ {δ_i}
6     else
7         Δ ← positiveDecisions ∪ {¬δ_i}
8         Γ ← Γ ∪ {Δ} ∪ computeSymmetricNogoodsOf(Δ)
9 return Γ
```

The function computeSymmetricNogoodsOf admits as a parameter a nogood Δ and returns a set of standard nogoods that are all symmetric to Δ. Note that this function uses three global data: $\mathscr{B}$, Ψ and *updateQueue*. $\mathscr{B}$ is the global nogood base. Ψ is a set of symmetries identified for the CN to be solved. For example, this set may only contain in practice the symmetries corresponding to the generators returned by a graph automorphism software. *updateQueue* is a Boolean that indicates whether a fixed point has to be reached or not, that is to say, whether all symmetrical nogoods have to be computed or not.

Algorithm 2: computeSymmetricNogoodsOf(Δ: nogood): set of nogoods

Global: a nogood base $\mathscr{B}$
Global: a set of symmetries Ψ of the CN to be solved
Global: a Boolean *updateQueue*
Input: a nogood Δ
Output: a set of nogoods Γ

```
1  Γ ← ∅
2  Q ← {Δ}
3  while Q ≠ ∅ do
4      pick and delete Δ' from Q
5      foreach symmetry σ of Ψ do
6          Δ'' ← Δ'^σ
7          if Δ' is admissible for σ and Δ'' ∉ ℬ then
8              Γ ← Γ ∪ {Δ''}
9              if updateQueue then
10                 Q ← Q ∪ {Δ''}
11 return Γ
```

A set Q is used to store nogoods from which symmetrical variants have to be sought. Initially, Q is initialized with the nogood Δ given as a parameter. Each nogood Δ' is successively picked and deleted from Q and symmetrical nogoods are computed. More precisely, a nogood Δ'' is computed from Δ' for each symmetry σ in Ψ. If Δ' is not admissible for σ (this may be checked from Δ'', for example) and if Δ'' is not already present in $\mathscr{B}$ (because already obtained using another symmetry, for example) then Δ'' is added to Γ. If a fixed point has to be reached (*updateQueue* set to *true*), Δ'' is added to Q in order to keep computing symmetrical nogoods from Δ'' in a next step. Notice that Δ' is necessarily admissible when Ψ only contains variable symmetries.

To illustrate symmetry-reinforced nogood recording from restarts, let us consider the 4-queens instance, and suppose that Ψ only contains the variable symmetry f^h. Figure 3 depicts the search tree built by FC (Forward Checking : a backtrack search algorithm maintaining a partial form of arc consistency at each node) after three decisions. Suppose that after the current decisions $x_a = 1$ and $x_b \neq 2$, the current run is stopped (due to the restart strategy). One can extract a nld-subsequence from the current branch : $\langle x_a = 1, x_b \neq 2\rangle$ and derive a nld-nogood $\Delta : \{x_a = 1, x_b = 2\}$. When f^h is applied on Δ, the symmetrical nogood $\Delta' = \{x_d = 1, x_c = 2\}$ is obtained (after swapping x_a with x_d and x_b with x_c). Suppose now that Ψ contains a second symmetry f^i. This means that f^h and f^i can be composed to generate more symmetrical nogoods (if *updateQueue* is set to *true*). For example, one can compute:

$$(\Delta)^{f^h}, \quad (\Delta)^{f^i}, \quad ((\Delta)^{f^i})^{f^h}, \quad ((\Delta)^{f^h})^{f^i}, \quad \ldots$$

until a fixed point is reached because no new symmetrical nogood can be generated.

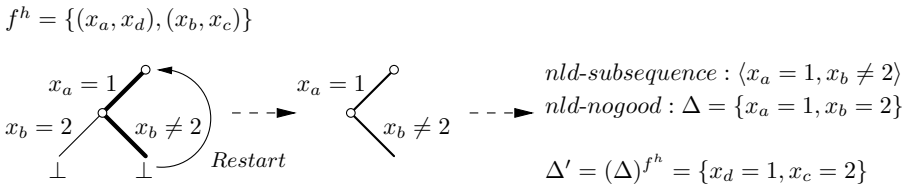


Fig. 3. Simple illustration of symmetry-reinforced nogood recording from restarts.

Classically, computing compositions of generators is not performed due to space and time explosion. Above, we have proposed to revisit indirectly compositions (when *updateQueue* is set to *true*) by iteratively computing symmetrical nogoods until a fixed point is reached. In our context, as the number of extracted nld-nogoods after each run is quite reasonable (polynomial wrt the number of variables and the greatest domain size), one can expect that the overhead will not be prohibitive.

6 Experiments

To show the practical value of symmetry-reinforced nogood recording from restarts, we have conducted an experimentation using a cluster of Xeon 3.0GHz with 1 GB of RAM under Linux. We have measured performance in terms of CPU time (in seconds) and the number of visited nodes. We have tested MAC (the backtrack search algorithm that maintains generalized arc consistency during search) with three variants of (symmetry-reinforced) nogood recording from restarts. The first variant, denoted by NG, is the classical nogood recording from restarts technique. The second variant, denoted by NG+Sym^1, is the technique of symmetry-reinforced nogood recording from restarts, Algorithm 1, when the Boolean *updateQueue* is set to $false$. That means that when a run is stopped, NG+Sym^1 exactly computes a single symmetrical nogood for each extracted nld-nogood and for each symmetry in Ψ. The third variant, denoted by NG+Sym*, is the technique of symmetry-reinforced nogood recording from restarts, Algorithm 1, when the Boolean *updateQueue* is set to *true*. That means that all possible symmetrical nogoods are computed from extracted nld-nogoods and symmetries in Ψ (i.e., the process is run until a fixed point is reached). We have also tested different variable ordering heuristics, namely *dom/ddeg* and *dom/wdeg* [3], and the time-out has been set to 20 minutes per instance. Finally, note that we have employed a geometric restart policy in which the number of backtracks is increased by a factor 1.5.

Instance		NG	NG+Sym^1	NG+Sym*
dom/ddeg	*#solved*	1,604	1,610	**1,618**
	cpu (1,602)	39.4	36.1	35.2
dom/wdeg	*#solved*	1,893	1,898	**1,909**
	cpu (1,889)	37.1	32.1	31.4

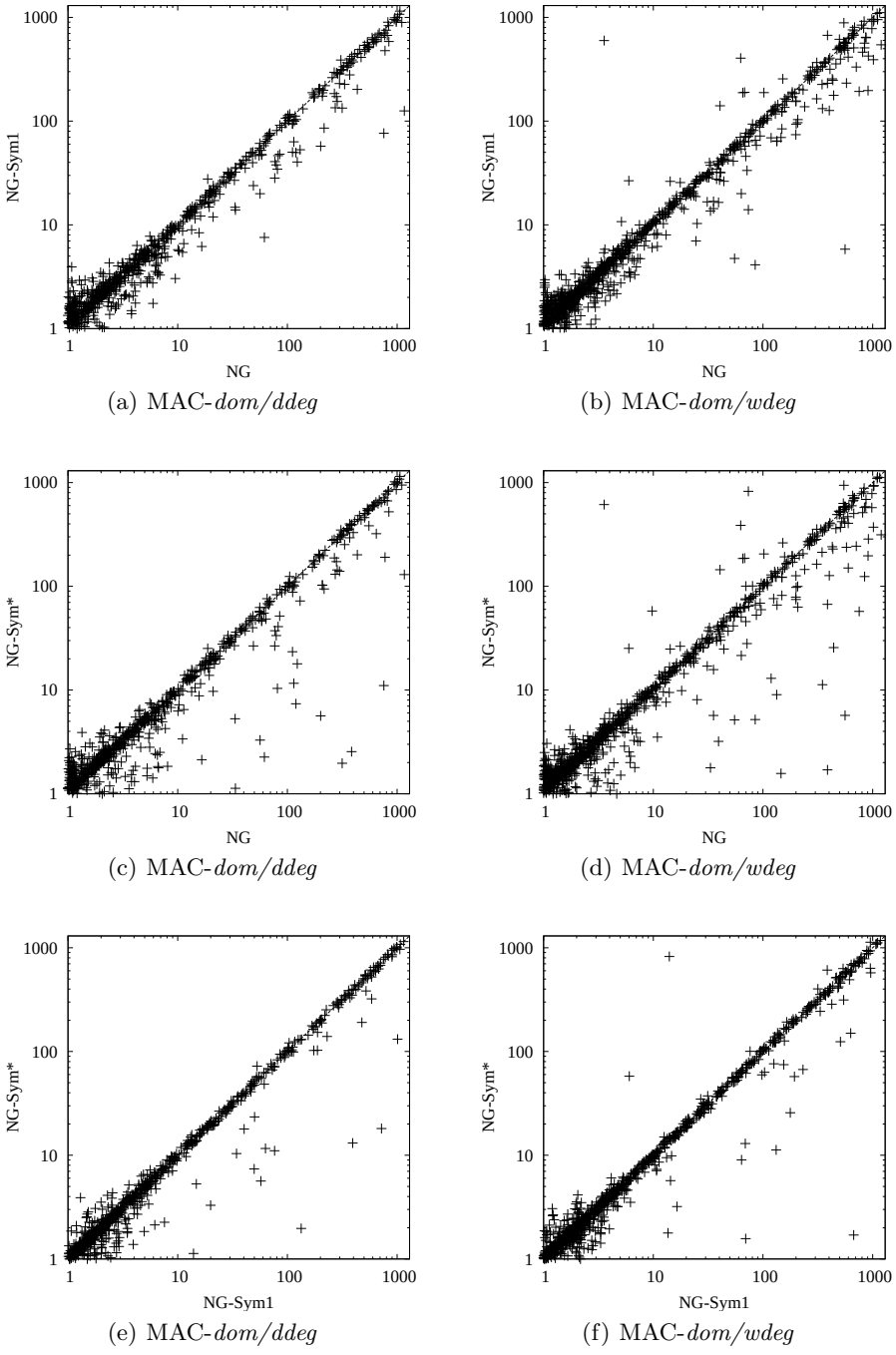
Table 1. Number of solved instances and average cpu time on a benchmark composed of 2,657 instances, given 20 minutes.

To identify automatically symmetries, we have used the lightweight detection technique of variable symmetries presented in [19]. For each instance, the generators (of the symmetry group) returned by Saucy [7] have been collected and recorded in the set Ψ. Note that the time spent by Saucy to compute these generators is quite negligible.

We have tested the three variants of MAC on 2,657 instances coming from `www.cril.fr/~lecoutre/benchmarks.html` after having selected all problems that contain variable symmetries (and that can be identified by our automatic lightweight detection technique). Table 1 provides an overview of the results in terms of the number of solved instances within the time limit. The average CPU time is computed from instances solved by the three methods. Note that the number of instances solved by the three methods is given into round brackets.

Instance		NG	NG+Sym[1]	NG+Sym*
2-insertions-3-3	cpu	4.36	2.33	**1.25**
#gen=2	nodes	48,975	20,238	7,375
3-insertions-3-3	cpu	387	231	**66.9**
#gen=2	nodes	$5,744K$	$3,191K$	$929K$
3-insertions-4-3	cpu	844	507	**124**
#gen=2	nodes	$10M$	$5,913K$	$1,397K$
fpga-10-10	cpu	908	197	**196**
#gen=26	nodes	$5,788K$	$1,251K$	$1,251K$
fpga-10-9	cpu	**151**	255	262
#gen=23	nodes	$1,187K$	$2,075K$	$2,075K$
fpga-12-10	cpu	time-out	**736**	738
#gen=28	nodes	-	$4,751K$	$4,751K$
graceful–K4-P2	cpu	1.13	**0.91**	1.37
#gen=4	nodes	1,924	708	503
graceful–K5-P2	cpu	599	630	**150**
#gen=4	nodes	$2,566K$	$2,573K$	$526K$
haystacks-04	cpu	0.48	**0.38**	0.48
#gen=8	nodes	328	149	114
haystacks-05	cpu	24.5	6.99	**0.89**
#gen=15	nodes	$305K$	74,951	2,451
haystacks-06	cpu	time-out	time-out	**869**
#gen=24	nodes	-	-	$10M$
pigeons-10	cpu	33.2	13.6	**1.78**
#gen=9	nodes	$456K$	$168K$	10,018
pigeons-11	cpu	348	131	**11.2**
#gen=10	nodes	$4,641K$	$1,716K$	$125K$
pigeons-12	cpu	time-out	time-out	**190**
#gen=11	nodes	-	-	$1,731K$
scen11-f1	cpu	time-out	951	**634**
#gen=38	nodes	-	$5,728K$	$3,729K$
scen11-f2	cpu	708	332	**243**
#gen=38	nodes	$4,154K$	$2,010K$	$1,352K$
scen11-f3	cpu	207	103	**63.0**
#gen=38	nodes	$1,269K$	$597K$	$353K$
series-14	cpu	63.4	**20.0**	21.5
#gen=1	nodes	$424K$	$130K$	$130K$
series-15	cpu	199	**73.9**	78.6
#gen=1	nodes	$1,210K$	$461K$	$461K$
series-16	cpu	time-out	432	**428**
#gen=1	nodes	-	$2,696K$	$2,696K$
val17-42	cpu	753	193	**57.3**
#gen=2	nodes	$1,944K$	$473K$	$134K$
val18-42	cpu	118	69.0	**12.9**
#gen=4	nodes	$247K$	$139K$	20,745
val18-44	cpu	75.0	75.1	**74.7**
#gen=4	nodes	$371K$	$371K$	$371K$

Table 2. Illustrative results obtained on some problem instances.



(a) MAC-*dom/ddeg* (b) MAC-*dom/wdeg*

(c) MAC-*dom/ddeg* (d) MAC-*dom/wdeg*

(e) MAC-*dom/ddeg* (f) MAC-*dom/wdeg*

Fig. 4. Pairwise comparisons (cpu time) on a benchmark composed of $2,657$ instances.

Whatever variable ordering heuristic is used, the number of solved instances increases when symmetry-reinforced nogood recording is used. Also, the variant NG+Sym* outperforms NG+Sym^1 both in terms of solved instances and CPU time. It is interesting to note that there is no significant overhead when NG+Sym* is used (in other words, this is not really penalizing to compute indirectly "composition" of symmetries). This is due to the fact that the number of extracted nld-nogoods is limited and a fixed point can be reached quickly in practice.

Table 2 focuses on some illustrative instances when the heuristic *dom/wdeg* is used. Clearly, for the hardest RLFAP instances, the symmetry-reinforced methods allow greater efficiency. For example, the two NG+Sym approaches solve the scen11-f1 instance while this instance remains unsolved by NG. The gap between NG+Sym^1 and NG+Sym* is less significant on some other series such as series. On well-known symmetrical problems such as haystack or pigeons (modeled as a clique of binary difference constraints), NG+Sym* outperforms (unsurprisingly) the other variants.

Finally, Figure 4 represents scatter plots displaying pairwise comparisons for NG, NG+Sym^1 and NG+Sym* when the heuristics *dom/ddeg* (subfigures on the left) and *dom/wdeg* (subfigures on the right) are used. Note the presence of many dots located under the diagonal line of subfigures 4(c) and 4(d), which represent instances solved quicker by the methods whose name labels the y-axis. Clearly, NG+Sym^1 and NG+Sym* outperform NG. When NG+Sym^1 and NG+Sym* are compared, it appears that NG+Sym* is slightly better.

7 Conclusion

In this paper, we have introduced the principle of symmetry-reinforced nogood recording from restarts. An original method is NG+Sym* that allows us to benefit indirectly from the composition of symmetries by iteratively computing symmetrical nogoods (until a fixed point is reached). It is worthwhile to recall that classically, only a small set of symmetries is used in practice (e.g., the generators of a symmetry group identified by a graph automorphism software), and no composition is computed. As a first perspective, we would like to study the practical interest of NG+Sym* with other kinds of symmetries such as value symmetries and variable-value symmetries. A second perspective is to compare our approach with classical symmetry-breaking methods.

References

1. R. Backofen and S. Will. Excluding symmetries in constraint based search. *Constraints*, 7(3-4):333–349, 2002.
2. B. Benhamou. Study of symmetry in constraint satisfaction problems. In *Proceedings of CP'94*, pages 246–254, 1994.
3. F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.

4. G. Chu, P. Stuckey, M. Garcia de la Banda, and C. Mears. Symmetries and lazy clause generation. In *Proceedings of IJCAI'11*, pages 516–521, 2011.
5. D. Cohen, P. Jeavons, C. Jefferson, K.E. Petrie, and B.M. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints*, 11(2-3):115–137, 2006.
6. J. Crawford, M.L. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proceedings of KR'96*, pages 148–159, 1996.
7. P.T. Darga, M.H. Liffiton, K.A. Sakallah, and I.L. Markov. Exploiting structure in symmetry generation for CNF. In *Proceedings of DAC'04*, pages 530–534, 2004.
8. R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.
9. T. Fahle, S. Schamberger, and M. Sellman. Symmetry breaking. In *Proceedings of CP'01*, pages 93–107, 2001.
10. P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *Proceedings of CP'02*, pages 462–476, 2002.
11. F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proceedings of CP'01*, pages 77–92, 2001.
12. I.P. Gent, W. Harwey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In *Proceedings of CP'02*, pages 415–430, 2002.
13. I.P. Gent and B.M. Smith. Symmetry breaking during search. In *Proceedings of ECAI'00*, pages 599–603, 2000.
14. C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, 2000.
15. D. Heller and M. Sellmann. Dynamic symmetry breaking restarted. In *Proceedings of CP'06*, pages 721–725, 2006.
16. G. Katsirelos and F. Bacchus. Generalized nogoods in CSPs. In *Proceedings of AAAI'05*, pages 390–396, 2005.
17. T. Kelsey, S. Linton, and C.M. Roney-Dougal. New developments in symmetry breaking in search using computational group theory. In *Proceedings of AISC'04*, pages 199–210, 2004.
18. C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Recording and minimizing nogoods from restarts. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 1:147–167, 2007.
19. C. Lecoutre and S. Tabary. Lightweight detection of variable symmetries for constraint satisfaction. In *Proceedings of ICTAI'09*, pages 193–197, 2009.
20. P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence*, 129(1-2):133–163, 2001.
21. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of DAC'01*, pages 530–535, 2001.
22. J.F. Puget. Automatic detection of variable and value symmetries. In *Proceedings of CP'05*, pages 475–489, 2005.
23. J.F. Puget. Symmetry breaking revisited. *Constraints*, 10(1):23–46, 2005.
24. M. Sellmann and P. van Hentenryck. Structural symmetry breaking. In *Proceedings of IJCAI'05*, pages 298–303, 2005.
25. S. Szeider. The complexity of resolution with generalized symmetry rules. *Theory of Computing Systems*, 38(2):171–188, 2005.
26. T. Walsh. General symmetry breaking constraints. In *Proceedings of CP'06*, pages 650–664, 2006.
27. L. Zhang and S. Malik. The quest for efficient Boolean satisfiability solvers. In *Proceedings of CADE'02*, pages 295–313, 2002.